UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/727,240	12/03/2003	Jose Abad Peiro	200313161-1	4907

22879 7590 02/09/2007

HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400

EXAMINER	
TRAN, QUOC A	
ART UNIT	**PAPER NUMBER**
2176	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	02/09/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)	
	10/727,240	PEIRO ET AL.	
	Examiner	Art Unit	
	Tran A. Quoc	2176	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *16 November 2006*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-34* is/are pending in the application.

4a) Of the above claim(s) *29-31* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-28, and 32-34* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-1

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is a final rejection in response to amendment filed on 11-16-2006.

2. Claims 1-34 are pending. Claims 29-31 are not elected.

3. Claims 1-17, 19-28, and 32-34 have been amended.

4. Claims 1, 9, 14, 21, 25, and 32-33 are independent claims.

5. Effective filing date 12-03-2003.

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> ***(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.***

6-1) Claims 1-10, 12-15, 17-22, and 24-28 rejected under 35 U.S.C. 103(a) as being unpatentable over Gebert et al (US 2002/0111963, published on 8/25/2006) (hereinafter Gebert, in view of Nonpatent Literature PODI Technical Meeting (Nov 2000, available at http://www.info-dist.com/lg/technical/argon/argon_public.pdf>)(hereinafter "PODI").

Regarding independent claim 1, Gebert teaches **the processor-executable instructions comprising instructions for: parsing structures within the PPML (Personalized Print Markup Language) (hereinafter PPML) document; generating a PDF (Portable Document Format) (hereinafter PDF) document tree; interpreting the parsed structures from the PPML document onto locations on the PDF document tree**. For example, Gebert discloses a method and program for preprocessing a document to render on an output device (Gebert, Title) where the program converts XML document into a viewable format such as PDF by using an

XML parser to parse the XML result tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7).

In addition, Gebert teaches **configuring a PDF document according to the PDF document tree.** For example, Gebert discloses a program that converts XML document into a viewable format such as PDF by using an XML parser to parse the XML result tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7) in order to transform the XML document for output on a printer (Gebert para 11).

In the broadest reasonable interpretation, Examiner equates **generating a PDF document from a PPML document** as equivalent to generating a PDF document from an XML document, because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML, and the claimed **PDF document tree** as equivalent to a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM as taught by Gebert.

Gebert does not expressly teach, but PODI teaches **generating a PDF document from a PPML document**. Specifically, a presentation at a Technical Meeting in 2000 by think121.com,

where PODI expressly disclosed product plug-ins for converting vendor and device-specific PPML to generic PDF (see presentation slides, slide page 30 as shown on page 20 of the attached reference).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert to include a plug-ins for converting PPML documents to PDF documents as expressly disclosed by PODI, providing the benefit of efficient and real-time manipulation of underlying PDF document and structures (PODI, slide 5) and providing the user tremendous flexibility in reuse of information (PODI, slide 25 and 26) for a printer (Gebert, para 11), which are similar benefits sought by the Applicant in the invention of allowing users to use lower-end printers that can process PDF or Postscript documents for print jobs that originated with PPML documents (see Applicant specification, Background section, page 1, para 2 and 3).

Regarding claim 2, Gebert teaches **printing the PDF document**. For example, Gebert discloses one or more page objects (22a...n) to an output, and allows a printer or some other device rasterizes to produce output (Gebert para 25). In addition, Gebert discloses the page objects (22a . . . n) may be expressed in other presentation languages, including Page Description Language (PDL) presentation languages, e.g., PDF, PostScript, etc. Page objects in a standard presentation document format may be rasterized by standard transforms known in the art (Gebert para 34).

Regarding claim 3, Gebert teaches **interpreting the parsed structures comprising: instructions for resolving, for objects within the PPML document, a PPML SOURCE_TYPE class; and translating the objects according to the PPML SOURCE_TYPE class**. For example, Gebert discloses processing a source XML document to

determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12). In the broadest reasonable interpretation, Examiner interprets Gebert's teaching as equivalent to the claimed functionality, which is to interpret the parsed structure, resolving the objects of the document and then translating the objects in order to prepare them for printing.

Regarding claim 4, Gebert teaches **un-marshalling a PPML instance, when a PPML tag refers to an external object; and embedding the external object into the PDF document**. Specifically, Gebert discloses an XML parser to parser XML document to parser XML result tree and then transform the tree elements into output in a PDF (Gebert para 8), where the transformation may contain text, graphic or images objects to define the page layout for presentation to a reader (Gebert para 7). In the broadest reasonable interpretation, Examiner interprets the claimed "**un-marshalling a PPML**" as equivalent to parsing because the specification describe un-marshalling as one of the steps for parsing (see Applicant specification, page 6, para 19).

Regarding claim 5, Gebert teaches **wherein the external object to which the PPML tag refers includes fonts and images**. For example, Gebert discloses nodes of the result tree including format objects such as fonts and images for rendering on a printer device (Gebert para 6 -7).

Regarding claim 6, Gebert teaches **interpreting the parsed structures comprises instructions for: resolving references within the parsed structures into PDF assets; and incorporating the PDF assets into the PDF document**. Specifically, Gebert discloses

processing a source XML document to determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12). Also, Gebert discloses the page objects (22a.... n), such as PostScript, PDF, XML formatted objects, etc, and stored or archived and then later rendered on different output devices in a manner that provides optimal output as the particular device rasterizer page objects according to its device specific rasterizer, wherein the printer driver 6 may use (Gebert para 36). In the broadest reasonable interpretation, Examiner equates the claimed **incorporating the PDF assets into the PDF document** PostScript, PDF, XML formatted objects, etc, and stored or archived and then later rendered on different output devices in a manner that provides optimal output as the particular device rasterizer page objects according to its device specific rasterizer as taught by Gebert.

Regarding claim 7, Gebert **teaches resolving an image asset into an InputSource object using a PPML structure; and incorporating the image asset into the PDF document.** For example Gebert discloses processing a source XML document to determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12) for optimal rendering of PDF documents at the printer nodes of the result tree including format objects such as fonts and images for rendering on a printer device (Gebert para 6 and 7).

Regarding claim 8, Gebert teaches **the PPML structure comprises:**

<SOURCE Format="image/jpeg" Dimensions "dim1 dim2">

<EXTERNAL_DATA_ARRAY Src="filename.jpg"/> </SOURCE>. In the broadest reasonable interpretation, the Examiner interprets this limitation as processing steps towards translating XML document into PDF document including external image objects Gebert discloses processing a source XML document to determine the document objects such as page divisions and formatting properties including image data objects (Gebert para 7), and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12) for optimal rendering of PDF documents at the printer nodes of the result tree including format objects such as fonts and images for rendering on a printer device (Gebert para 6 and 7).

Regarding independent claim 9, Gebert teaches **instructions for resolving assets within the parsed structures into objects; and incorporating the objects into the PDF document**. For example, Gebert discloses processing a source XML document to determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12) for optimal rendering of PDF documents at the printer.

Regarding claim 10, Gebert teaches **resolving assets using a PPML structure to translate an asset into an iText font representing a TTF font**. In the broadest reasonable interpretation, Examiner reads this claim as using an XML structure to translate fonts in order to have them retain properties for an accurate representation when generating the fonts in a PDF. Specifically, Gebert discloses using an XML structure (a DOM tree) of nodes for elements of XML source documents to construct the result tree for transformation of objects including fonts (Gebert para 6) and for then transforming the tree elements into PDF (Gebert para 8).

Regarding claim 12, Gebert teaches **interpreting the parsed structures comprises instructions for: locating PPML global impositions when parsing the structures within the PPML document; and reutilizing the PPML global impositions**. In the broadest reasonable interpretation, Examiner reads the claimed impositions as properties for a template that define how logical pages will be mapped into physical pages and may be input as parameters by user (see Applicant's specification, page 15, para 47). Specifically, Gebert discloses classes of formatting objects and formatting properties (Gebert para 12) for transforming XML documents to output that is rasterized to a PDF (Gebert para 7). The XSL formatting objects generates from source XML document and defines the page layout for presentation to a reader (Gebert para 12). The examiner interprets this as reutilized because the source objects and nodes are reused for creating a resultant node tree or layout.

Regarding claim 13, Gebert does not expressly teach **instructions for reutilizing IMPOSITION_TYPE instances**, but does suggest it because Gebert discloses classes of formatting objects and formatting properties (Gebert para 12) for transforming XML documents to output that is rasterized to a PDF (Gebert para 7). The XSL formatting objects generates from source XML document and defines the page layout for presentation to a reader (Gebert para 12). It would have been obvious to one of ordinary skill in the art at the time of the invention to interpret Gebert's teaching of formatting objects and formatting properties for transforming XML documents to output that is rasterized to a PDF as equivalent to the claimed reutilizing instance because the source objects and nodes are reused for creating a resultant node tree or layout and this limitation would be an obvious step in the implementation of the transformation and layout process.

Regarding independent claim 14, Gebert teaches **a first component, operable on a processor, to parse structures and tag items within the PPML document; a second component to, operable on a processor, generate a PDF document tree; and a third component, operable on a processor, to translate the tagged items within the PPML document, and to locate the translated items on the PDF document tree**. For example, Gebert discloses a method and program for preprocessing a document to render on an output device (Gebert, Title) where the program converts XML document into a viewable format such as PDF by using an XML parser to parse the XML result tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7). In the broadest reasonable interpretation, Examiner reads the claimed "**generating a PDF document from a PPML document**" as equivalent to generating a PDF document from an XML document, because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML.

Gebert does not expressly teach, "**Converting a PPML document into a PDF document**", but PODI does teach this limitation. Specifically, PODI discloses a presentation at a Technical Meeting in 2000 by think121.com, where PODI expressly disclosed product plug-ins for converting vendor and device-specific PPML to generic PDF (see presentation slides, slide page 30 as shown on page 20 of the attached reference).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert to include a plug-ins for converting PPML documents to PDF documents as expressly disclosed by PODI, providing the benefit of efficient and real-time manipulation of underlying PDF document and structures (PODI, slide 5) and providing the user tremendous flexibility in reuse of information (PODI, slide 25 and 26) for a printer (Gebert, para 11), which are similar benefits sought by the Applicant in the invention of allowing users to use lower-end printers that can process PDF or Postscript documents for print jobs that originated with PPML documents (see Applicant specification, Background section, page 1, para 2 and 3).

Regarding claim 15, Gebert teaches **a forth component to un-marshal PPML instances of external objects for embedding within the PDF document tree**. For example, Gebert discloses an XML parser to parser XML document to parser XML result tree and then transform the tree elements into output in a PDF (Gebert para 8), where the transformation may contain text, graphic or images objects to define the page layout for presentation to a reader (para 7). In the broadest reasonable interpretation, Examiner interprets the claimed "**un-marshalling a PPML**" as equivalent to parsing because the specification describe un-marshalling as one of the steps for parsing (see Applicant specification, page 6, para 19).

Regarding claim 17, Gebert teaches **resolving PPML SOURCE_TYPE class of the structures within the PPML document**. For example, Gebert discloses processing a source XML document to determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12). In the broadest reasonable interpretation, Examiner reads Gebert's teaching as equivalent to the claimed

functionality, which is to interpret the parsed structure, resolving the objects of the document and then translating the objects in order to prepare them for printing.

Regarding claim 18, Gebert teaches **the component translating the items tagged as a function of a type associated with each of the items tagged**. For example, Gebert teaches transforming source XML DOM tree into a result DOM tree structure to allowing printing PDF documents, where the source tree and result tree have classes of formatting objects information such as page, paragraph, table, font, etc., (Gebert para 6). In the broadest reasonable interpretation, examiner interprets these classes of formatting information as equivalent to the claimed tagged items of a type, because the Examiner interprets tags as details about objects in order to classify the data object into a category/class for transforming a particular class and Gebert's formatting objects such as fonts can be of a font type of formatting object information.

Regarding claim 19, Gebert teaches **translating a PDF object within a PdfTemplate defined as a function of an item type of the tagged items found during parsing by the first component**. For example, Gebert teaches transforming source XML DOM tree into a result DOM tree structure to allowing printing PDF documents, where the source tree and result tree have classes of formatting objects information such as page, paragraph, table, font, etc., (Gebert para 6). In the broadest reasonable interpretation, Examiner interprets these classes of formatting information as equivalent to the claimed **tagged items of a type**, because the Examiner interprets tags as details about objects in order to classify the data object into a category/class for transforming a particular class and Gebert's formatting objects such as fonts can be of a font type of formatting object information. Additionally, Gebert discloses the XML parser to parse the XML formatting objects result tree and then transform the tree elements into a PDF document.

Regarding claim 20, Gebert teaches **wherein the PdfTemplate is configured to support caching of objects to optimize PDF structure**. Fro example, Gebert discloses optimizing output for rendering on a specific printer (Gebert para 10).

Regarding independent claim 21, Gebert teaches **means for parsing structures within the PPML document; means for generating a PDF document tree; and means for interpreting the parsed structures from the PPML document onto locations on the PDF document tree; wherein the means for parsing, means for generating and the means for interpreting are performed impart by operation of processor**. For example, Gebert discloses a method and program for preprocessing a document to render on an output device (Gebert, Title) where the program converts XML document into a viewable format such as PDF by using an XML parser to parse the XML result tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7). In the broadest reasonable interpretation, Examiner interprets the claimed "**generating a PDF document from a PPML document**" as equivalent to generating a PDF document from an XML document, because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML.

In addition, Gebert does not expressly teach, "**converting a PPML document into a PDF document**", but PODI does teach this limitations. Specifically, PODI discloses a

presentation at a Technical Meeting in 2000 by think121.com, where PODI expressly disclosed product plug-ins for converting vendor and device-specific PPML to generic PDF (see presentation slides, slide page 30 as shown on page 20 of the attached reference).
It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert to include a plug-ins for converting PPML documents to PDF documents as expressly disclosed by PODI, providing the benefit of efficient and real-time manipulation of underlying PDF document and structures (PODI, slide 5) and providing the user tremendous flexibility in reuse of information (PODI, slide 25 and 26) for a printer (Gebert, para 11), which are similar benefits sought by the Applicant in the invention of allowing users to use lower-end printers that can process PDF or Postscript documents for print jobs that originated with PPML documents (see Applicant specification, Background section, page 1, para 2 and 3).

Regarding claim 22, Gebert teaches **means for un-marshalling a PPML instance, when a PPML tag set by the means for parsing refers to an external object; and means for embedding the external object into the PDF document**. Specifically, Gebert discloses an XML parser to parser XML document to parser XML result tree and then transform the tree elements into output in a PDF (Gebert para 8), where the transformation may contain text, graphic or images objects to define the page layout for presentation to a reader (Gebert para 7). In the broadest reasonable interpretation, Examiner interprets the claimed "**un-marshalling a PPML**" as equivalent to parsing because the specification describe un-marshalling as one of the steps for parsing (see Applicant specification, page 6, para 19).

Regarding claim 24, Gebert teaches **means for resolving, for objects within the PPML document, a PPML SOURCE_TYPE class; and means for translating the objects**

according to the PPML SOURCE_TYPE class. For example, Gebert discloses processing a source XML document to determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12). In the broadest reasonable interpretation, Examiner equates Gebert's teaching as equivalent to the claimed functionality, which is to interpret the parsed structure, resolving the objects of the document and then translating the objects in order to prepare them for printing.

Regarding independent claim 25, Gebert teaches **parsing structures within the PPML document; generating a PDF document tree; and interpreting the parsed structures from the PPML document onto locations on the PDF document tree, wherein the means for parsing, means for generating and the means for interpreting are performed impart by operation of processor, which generating a PDF document tree**. For example, Gebert discloses a method and program for preprocessing a document to render on an output device (Gebert, Title) where the program converts XML document into a viewable format such as PDF by using an XML parser to parse the XML result tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7). In the broadest reasonable interpretation, Examiner interprets the claimed "**generating a PDF document from a PPML document**" as equivalent to generating a PDF document from an XML document, because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2),

thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML.

Gebert does not expressly teach, "**Converting a PPML document into a PDF document**", but PODI does teach these limitations. Specifically, PODI discloses a presentation at a Technical Meeting in 2000 by think121.com, where PODI expressly disclosed product plug-ins for converting vendor and device-specific PPML to generic PDF (see presentation slides, slide page 30 as shown on page 20 of the attached reference).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert to include a plug-ins for converting PPML documents to PDF documents as expressly disclosed by PODI, providing the benefit of efficient and real-time manipulation of underlying PDF document and structures (PODI, slide 5) and providing the user tremendous flexibility in reuse of information (PODI, slide 25 and 26) for a printer (Gebert, para 11), which are similar benefits sought by the Applicant in the invention of allowing users to use lower-end printers that can process PDF or Postscript documents for print jobs that originated with PPML documents (see Applicant specification, Background section, page 1, para 2 and 3).

Regarding claim 26, Gebert teaches **means for un-marshalling a PPML instance, when a PPML tag set during parsing refers to an external object; and embedding the external object into the PDF document**. Specifically, Gebert discloses an XML parser to parser XML document to parser XML result tree and then transform the tree elements into output in a PDF (Gebert para 8), where the transformation may contain text, graphic or images objects to define the page layout for presentation to a reader (Gebert para 7). In the broadest reasonable interpretation, Examiner interprets the claimed "**un-marshalling a PPML**" as equivalent to

parsing because the specification describe un-marshalling as one of the steps for parsing (see Applicant specification, page 6, para 19).

Regarding claim 27, Gebert teaches **interpreting the parsed structures comprises instructions for: resolving references within the parsed structures into PDF assets; and incorporating the PDF assets into the PDF document**. For example, Gebert discloses processing a source XML document to determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12) for optimal rendering of PDF documents at the printer.

Regarding claim 28, Gebert teaches **resolving an image asset into an InputSource object using a PPML structure; and incorporating the image asset into the PDF document**. For example, Gebert discloses processing a source XML document to determine the document objects such as page divisions and formatting properties, and then transmitting the page objects to transform the page objects into a readable information capable of being generated by an output device (Gebert para 12) for optimal rendering of PDF documents at the printer nodes of the result tree including format objects such as fonts and images for rendering on a printer device (Gebert para 6 and 7).

6-2) Claims 11, 16 and 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gebert (as cited above), in view of PODI (as cited above), further in view of Nonpatent Literature "The PPML Print Language in XML Workflows for Digital Print" (May 2001 by Dave deBronkart, available at <http://www.gca.org/papers/xmleurope2001/papers/html/sid-03-5.html>)(hereinafter "deBronkart").

Regarding claim 11, Gebert in view of PODI does not teach, but deBronkart teaches **interpreting the parsed structures comprises instructions for reutilizing PPML global objects found within the PPML document with a PPML structure, wherein the PPML global objects are configured as REUSABLE_OBJECT_TYPE**. Specifically, DeBronkart discloses a PPML print language in XML workflow for digital print (deBronkart ,Title), where personalized documents with reusable content are largely mapped onto conventional workflows wherein the variable data records trigger the selection of reusable objects, using logic rules embedded in a template for generating PDF content objects using a PDF generating tool (deBronkart page 3, section 3.3).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert in view of PODI to include reusable content objects are mapped to conventional workflows with variable data records for generating PDF content objects from PPML objects as taught by deBronkart, providing the benefit of promoting interoperability amongst printers allowing users to have a fast time to market (deBronkart, page 3, section 4) for variable data printing applications, allowing users to use desktop printers (deBronkart, Abstract), which is similar to the goal of the Applicant's invention of allowing users to use any printer such as low-

end printers instead of high-end digital presses (see Applicant's invention specification, page 1, para 3).

Regarding claim 16, Gebert teaches **a SourceResolver to resolve references into assets for attachment to the PDF document tree**. In the broadest reasonable interpretation, Examiner interprets this limitation as a system to resolve objects with the PDF document tree, including image objects. Specifically, Gebert discloses an XML parser to parser XML document to parser XML result tree and then transform the tree elements into output in a PDF (Gebert para 8), where the transformation may contain text, graphic or images objects to define the page layout for presentation to a reader (Gebert para 7). The result document (i.e., PDF document) includes root elements and page sequence elements such as text and images (Gebert para 24). The resultant PDF document is in a DOM tree of nodes of elements with rearranged source data objects and added information necessary to format the result tree for presentation, upon completion of a tree transformation (Gebert para 6).

In addition Gebert teaches **a FontResolver Interface to resolve fonts for access by the PDF document**, and a result PDF document from the transformation of the source XML document with nodes of element objects including fonts (Gebert para 6). The fonts in the XML source document are transformed over to the resultant PDF document.

Also, Gebert teaches an **ImpositioningStore to reutilize PPML global impositions for access by the PDF document**. In the broadest reasonable interpretation, Examiner interprets the limitation of impositions as properties for a template that define how logical pages will be mapped into physical pages and may be input as parameters by user (see Applicant's specification, page 15, para 47). Also, Gebert discloses classes of formatting objects and

formatting properties (Gebert para 12) for transforming XML documents to output that is rasterized to a PDF (Gebert para 7). And, the XSL formatting objects generates from source XML document and defines the page layout for presentation to a reader (Gebert para 7 and 12). In the broadest reasonable interpretation, Examiner interprets this as reutilized because the source objects and nodes are reused for creating a resultant node tree or layout.

Gebert in view of PODI does not teach, but deBronkart suggests **an OccurrenceStore Interface to reutilize PPML global objects for access by the PDF document**. For example, DeBronkart discloses a PPML print language in XML workflow for digital print (deBronkart ,Title), where personalized documents with reusable content are largely mapped onto conventional workflows wherein the variable data records trigger the selection of reusable objects, using logic rules embedded in a template for generating PDF content objects using a PDF generating tool (deBronkart page 3, section 3.3).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert in view of PODI to include reusable content objects are mapped to conventional workflows with variable data records for generating PDF content objects from PPML objects as taught by deBronkart, providing the benefit of promoting interoperability amongst printers allowing users to have a fast time to market (deBronkart, page 3, section 4) for variable data printing applications, allowing users to use desktop printers (deBronkart, Abstract), which is similar to the goal of the Applicant's invention of allowing users to use any printer such as low-end printers instead of high-end digital presses (see Applicant's invention specification, page 1, para 3).

Regarding claim 23, Gebert teaches **a means for resolving references into assets for attachment to the PDF document tree**. In the broadest reasonable interpretation, Examiner interprets this limitation as a system to resolve objects with the PDF document tree, including image objects. For example, Gebert discloses an XML parser to parser XML document to parser XML result tree and then transform the tree elements into output in a PDF (Gebert para 8), where the transformation may contain text, graphic or images objects to define the page layout for presentation to a reader (Gebert para 7). The result document (i.e., PDF document) includes root elements and page sequence elements such as text and images (para 24). The resultant PDF document is in a DOM tree of nodes of elements with rearranged source data objects and added information necessary to format the result tree for presentation, upon completion of a tree transformation (Gebert para 6).

In addition, Gebert teaches **a means for resolving fonts for access by the PDF document**. For example, Gebert discloses a result PDF document from the transformation of the source XML document with nodes of element objects including fonts (Gebert para 6). The fonts in the XML source document are transformed over to the resultant PDF document.

Also, Gebert teaches **means for reutilizing PPML global impositions for access by the PDF document**. In the broadest reasonable interpretation, Examiner interprets the limitation of impositions as properties for a template that define how logical pages will be mapped into physical pages and may be input as parameters by user (see Applicant's specification, page 15, para 47). Gebert discloses classes of formatting objects and formatting properties (Gebert para 12) for transforming XML documents to output that is rasterized to a PDF (Gebert para 7). The XSL formatting objects generates from source XML document and defines the page layout for

presentation to a reader (Gebert para 7 and 12). The examiner interprets this as reutilized because the source objects and nodes are reused for creating a resultant node tree or layout.

Gebert in view of PODI does not teach, but deBronkart suggests **means for reutilizing PPML global objects for access by the PDF document**. Specifically DeBronkart discloses a PPML print language in XML workflow for digital print (deBronkart, Title), where personalized documents with reusable content are largely mapped onto conventional workflows wherein the variable data records trigger the selection of reusable objects, using logic rules embedded in a template for generating PDF content objects using a PDF generating tool (deBronkart page 3, section 3.3).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert in view of PODI to include reusable content objects are mapped to conventional workflows with variable data records for generating PDF content objects from PPML objects as taught by deBronkart, providing the benefit of promoting interoperability amongst printers allowing users to have a fast time to market (deBronkart, page 3, section 4) for variable data printing applications, allowing users to use desktop printers (deBronkart, Abstract), which is similar to the goal of the Applicant's invention of allowing users to use any printer such as low-end printers instead of high-end digital presses (see Applicant's invention specification, page 1, para 3).

6-3) Claims 32-34 rejected under 35 U.S.C. 103(a) as being unpatentable over Gebert et al (US 2002/0111963, published on 8/25/2006), in view of PODi- "introduction to the Personalized Print Markup: The PPML Family of XML Standards" July, 2003 (hereinafter PODi). Further in view of Hardy et al. "Mapping and displaying structural transformation between XML and PDF" Published Nov, 2002 (hereinafter Hardy).

Regarding independent claim 32, Gebert teaches **a method for printing a PDF file, comprising: sending a PPML file to a PPML to PDF converter, wherein the converting is performed by operation of a processor, and printing the PDF file**. For example, Gebert discloses one or more page objects 22a . . . n to output that the printer or some other device rasterizes to produce output (Gebert para 25), where the page objects 22a . . . n may be expressed in other presentation languages, including page description language (PDL) presentation languages, e.g., PDF, PostScript, etc. Page objects in a standard presentation document format may be rasterized by standard transforms known in the art (Gebert para 34). In addition, Gebert discloses the program converts XML document into a viewable format such as PDF by using an XML parser to parse the XML result tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7). In the broadest reasonable interpretation, Examiner interprets the claimed **PPML** as equivalent to XML file as taught by Gebert as discuss above, and because PPML is an XML-based language for variable-data printing (see Applicant's specification

section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML.

In addition, Gebert does not explicitly teach, but PODi teaches **converting a PPML document into a PDF document**. Specifically, PODi discloses PPML is based on XML, the Extensible Markup Language, which has quickly become the universal syntax for data exchange (PODi page 1 Overview Section). Also, PODi discloses PPML is a simple, human-readable language that describes a document stream as a hierarchy of structured data. This example shows the basic structure of a PPML print stream that might be emitted by a PPML print driver or a PPML-enabled Web application, wherein each "Mark" element contains (or references) page content in any print language supported by the machine: PostScript, PDF, SVG, image formats such as JPEG, etc. This open, flexible architecture lets PPML be adapted to any sort of print environment, wherever vendors and users see an opportunity

```
<PPML...>...
    <DOCUMENT_SET...>...
        <DOCUMENT....>...
            <PAGE...>...
                <MARK...>....</MARK>
                <MARK...>....</MARK>
            </PAGE>
            <PAGE...>...</PAGE>
            ...
    </DOCUMENT>
    <DOCUMENT...>
```

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert to include a plug-ins for converting PPML documents to PDF documents as expressly disclosed by PODI, providing the benefit of efficient file size, resource management,

run printer at their full rated speed- even with graphics, enabling relevant document, and automated document workflows (PODi pages 3-4).

In addition, Gebert and PODi do not explicitly teach, but Hardy teaches **parsing and tagging items within the PPML file and translating the tagged items; generating a PDF document tree; interpreting the parsed structures from the PPML document onto locations on the PDF document tree.** Specifically, Hardy discloses a method of mapping and displaying structural transformation between XML and PDF (Hardy the Title), wherein it loads the XML file is loaded into a Document Object Model (DOM) parser. The DOM is a model for representing serialized XML trees as true trees. The plug-in creates a dialogue containing two tree-views. The left tree-view contains a representation of the PDF's internal structure tree, taken from the DOM and the right tree-view shows the structure and content of the XML starting document. Selecting a specific node in either tree gives access to the corresponding content in the opposite document (Hardy section 4.1, also fig. 1, fig. 4).

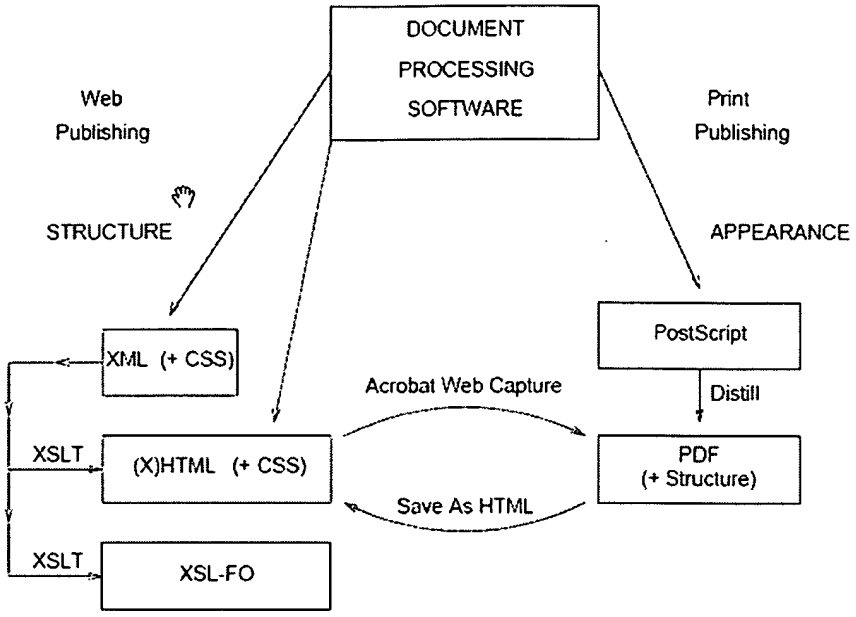

Figure 1. Document Processing Software.

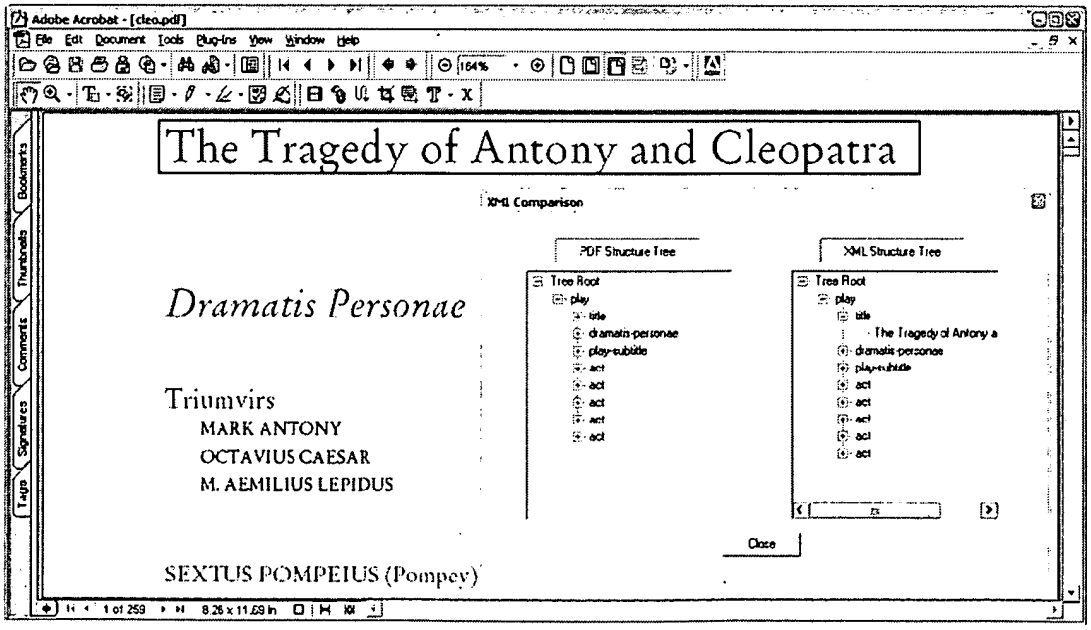

Figure 4. XML Comparison Plugin Usage

In the broadest reasonable interpretation, Examiner equates the claimed **generating a PDF document from a PPML document** as equivalent to generating a PDF document from an XML document, because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML as taught by Gebert and Hardy.

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gebert to include a plug-ins for converting PPML documents to PDF documents as expressly disclosed by PODI, and further to include a means for parsing and tagging items within the PPML file and translating the tagged items; generating a PDF document tree; interpreting the parsed structures from the PPML document onto locations on the PDF document tree as taught by Hardy. One of ordinary skill in the art would have been motivated to modify this combination to provide the benefit of efficient file size, resource management, run printer at their full rated speed- even with graphics, enabling relevant document, and automated document workflows (PODi pages 3-4).

Regarding independent claim 33, the rejection of claim 32 is fully incorporated.

Regarding claim 34, the rejection of claim 32 is fully incorporated. In addition, Gebert teaches **a translation component, to translate the assets located by the references; and a PDF tree-generating component, to generate a PDF tree to form the PDF file according to the translated assets**. For example, Gebert discloses a method and program for preprocessing a document to render on an output device (Gebert, Title) where the program converts XML document into a viewable format such as PDF by using an XML parser to parse the XML result

tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7). In the broadest reasonable interpretation, Examiner interprets the claimed "**parsing the PPML file**" as equivalent to XML parser to parse the XML result tree and then transform the tree elements into a PDF format as taught by Gebert, and because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2). Also, Examiner equates the claimed **a PDF tree-generating component, to generate a PDF tree to form the PDF file according to the translated assets** as equivalent to parse the XML result tree and then transform the tree elements into a PDF format as taught by Gebert, and because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2).

Response to Argument

7. Beginning on page 15 of 18 of the REMARKS (hereinafter the remarks), to address the amended portions of the currently amended claims 32-34, the Examiner introduces the PODi and Hardy references (see above rejection for details). As for the un-amended portions, Applicant argues the following issues, which are accordingly addressed below.

Applicant argues that the combination Gebert, PODI, and deBronkart fail to teach PPML to PDF translation (the remarks pages 15-16). The examiner disagrees, in the broadest reasonable interpretation, Examiner equates generating a PDF document from a PPML document as equivalent to generating a PDF document from an XML document, because PPML is an

XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML, and the claimed **PDF document tree** as equivalent to a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM as taught by Gebert.

Also, PODi discloses PPML is based on XML, the Extensible Markup Language, which has quickly become the universal syntax for data exchange (PODi page 1 Overview Section). Also, PODi discloses PPML is a simple, human-readable language that describes a document stream as a hierarchy of structured data. This example shows the basic structure of a PPML print stream that might be emitted by a PPML print driver or a PPML-enabled Web application, wherein each "Mark" element contains (or references) page content in any print language supported by the machine: PostScript, PDF, SVG, image formats such as JPEG, etc. This open, flexible architecture lets PPML be adapted to any sort of print environment, wherever vendors and users see an opportunity

```
<PPML...>...
    <DOCUMENT_SET...>...
        <DOCUMENT....>...
            <PAGE...>...
                <MARK...>....</MARK>
                <MARK...>....</MARK>
            </PAGE>
            <PAGE...>...</PAGE>
            ...
    </DOCUMENT>
    <DOCUMENT...>
```

And Furthermore, Hardy discloses a method of mapping and displaying structural transformation between XML and PDF (Hardy the Title), wherein it loads the XML file is loaded

into a Document Object Model (DOM) parser. The DOM is a model for representing serialized XML trees as true trees. The plug-in creates a dialogue containing two tree-views. The left tree-view contains a representation of the PDF's internal structure tree, taken from the DOM and the right tree-view shows the structure and content of the XML starting document. Selecting a specific node in either tree gives access to the corresponding content in the opposite document (Hardy section 4.1, also fig. 1, fig. 4).

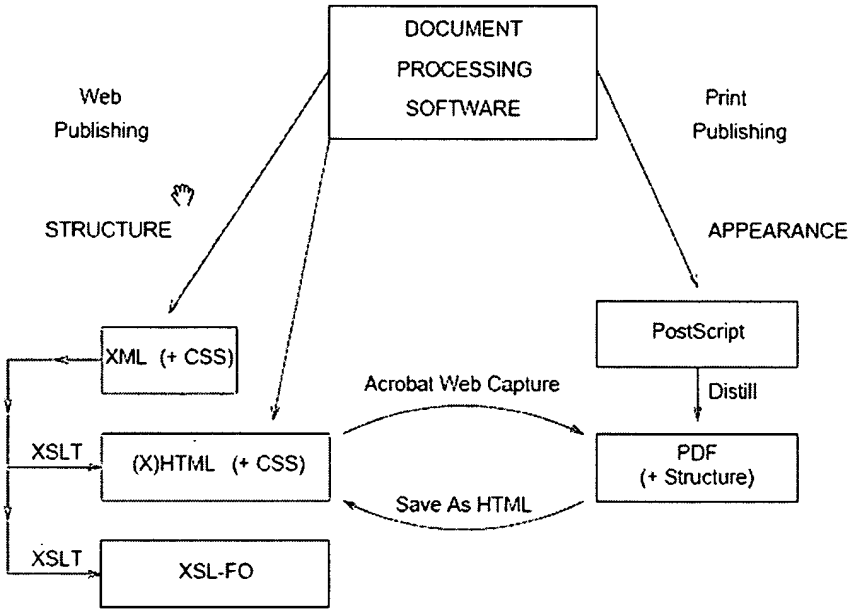

Figure 1. Document Processing Software.

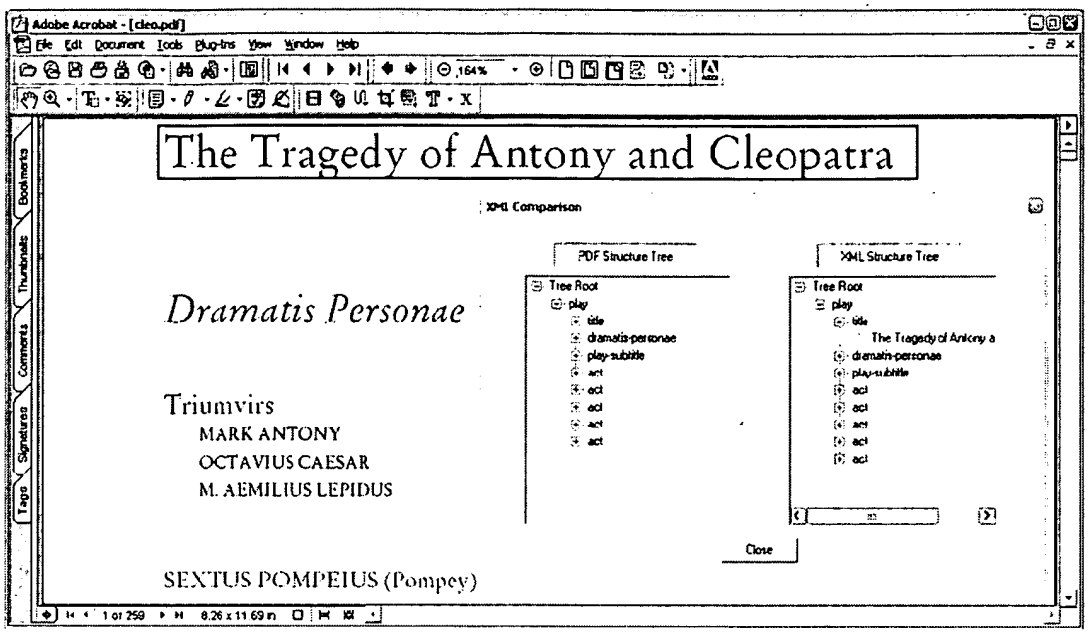

Figure 4. XML Comparison Plugin Usage

In the broadest reasonable interpretation, Examiner equates the claimed **generating a PDF document from a PPML document** as equivalent to generating a PDF document from an XML document, because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML as taught by Gebert and Hardy.

In addition, **Applicant argues that the combination Gebert, PODI, and deBronkart fail to teach generating (the remarks pages 16-18).** The examiner disagrees, Gebert discloses a program that converts XML document into a viewable format such as PDF by using an XML parser to parse the XML result tree and then transform the tree elements into a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM (Gebert para 8). Additionally, to transform the XML document, the entire source file is

transformed to a result tree such as formats objects that define an internal representation of the page layouts (Gebert para 7) in order to transform the XML document for output on a printer (Gebert para 11). In the broadest reasonable interpretation, Examiner equates **generating a PDF document from a PPML document** as equivalent to generating a PDF document from an XML document, because PPML is an XML-based language for variable-data printing (see Applicant's specification section, page 1, para 2), thus generating a PDF document from an PPML document is interpreted as requiring the same functionalities as generating a PDF from XML, and the claimed **PDF document tree** as equivalent to a PDF format including formatting object into DOM documents to access elements of the result tree in the DOM as taught by Gebert.

Also, Hardy discloses a method of mapping and displaying structural transformation between XML and PDF (Hardy the Title), wherein it loads the XML file is loaded into a Document Object Model (DOM) parser. The DOM is a model for representing serialized XML trees as true trees. The plug-in creates a dialogue containing two tree-views. The left tree-view contains a representation of the PDF's internal structure tree, taken from the DOM and the right tree-view shows the structure and content of the XML starting document. Selecting a specific node in either tree gives access to the corresponding content in the opposite document (Hardy section 4.1, also fig. 4).

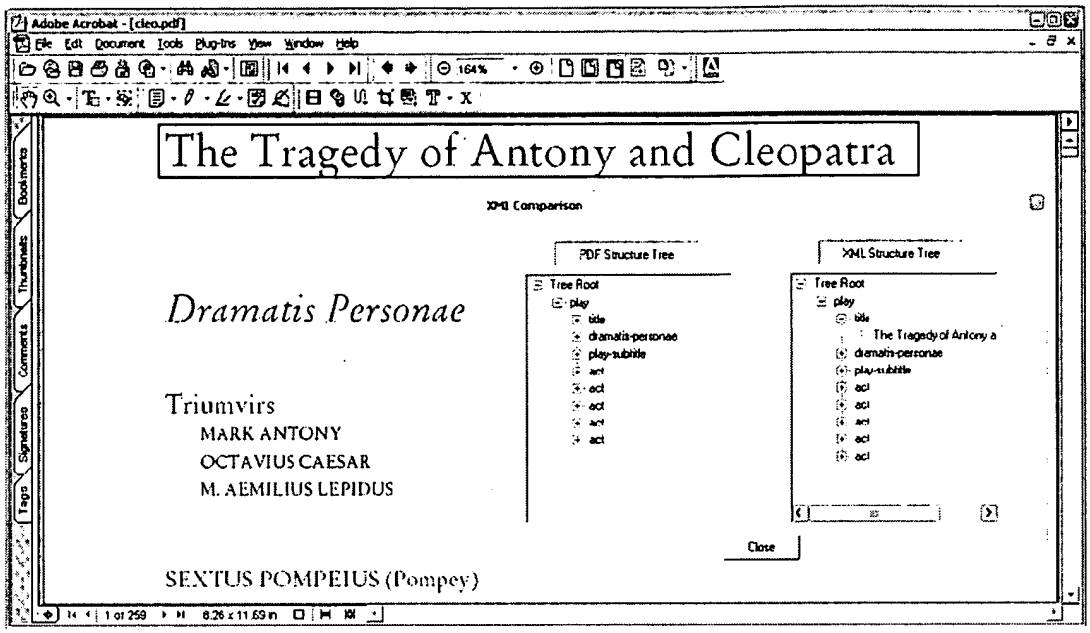

Figure 4. XML Comparison Plugin Usage

For at least all the above evidence, therefore the Examiner respectfully maintains the rejection of claims **1-28, and 32-34** should be sustained at this time.

Conclusion

8. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event,

however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Quoc A. Tran whose telephone number is 571-272-8664. The examiner can normally be reached on Monday through Friday from 9 AM to 5 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Herndon R. Heather can be reached on 571-272-4136. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Quoc A. Tran
Patent Examiner
2007-01-30

H. Herndon
Heather R. Herndon
Supervisory Patent Examiner
Technology Center 2100